กราฟิกไลบรารี บน Turbo C เวอร์ชัน 1.5

สมเกียรติ เอกจริยาวัฒน์

ในตอนที่บริษัท บอร์แลนด์ ได้ออก C Compiler สู่ตลาดที่มีชื่อว่าเทอร์โบซี ตามสไตล์ของบอร์แลนด์ ซึ่งนับว่าเป็นคอมพายเลอร์ที่มีความเร็วในการคอม-พายล์ และลิงค์สูง

เวอร์ชันแรกของเทอร์โบซียังไม่มีไลบรารีเกี่ยวกับกราฟิกให้ใช้ แต่ต่อมาบอร์แลนด์ก็ได้ออกเทอร์โบซีเวอร์ชัน 1.5 ซึ่งปรับปรุงส่วนต่าง ๆ และโดยเฉพาะอย่างยิ่งได้มีไลบรารีทางกราฟิก และเวอร์ชันนี้ ได้ออกมาไล่ ๆ กับเทอร์โบปาสคาลเวอร์ชัน 4 และเมื่อลองเปรียบเทียบกราฟิกไลบรารีบนเทอร์โบซีกับกราฟิกยูนิตบนเทอร์โบปาสคาลเวอร์ชัน 4 แล้ว เรียกได้ว่าเหมือนกันเลยทีเดียว ก็แสดงว่าถ้าเราคุ้นเคยฟังก์ชันทางกราฟิกของตัวใดตัวหนึ่งเราก็จะใช้อีกตัวหนึ่งได้อย่างง่ายดาย เนื่องจากทั้งชื่อและการส่งผ่านตัวแปรเหมือนกัน เพียงแต่เปลี่ยนตามรูปแบบของภาษาเท่านั้นเอง และในบทความนี้จะกล่าวถึงฟังก์ชันทางกราฟิกในเทอร์โบซี ซึ่งแบ่งเป็นกลุ่มคำสั่งใหญ่ดังแสดงในรูปที่ 1

รูปที่ 1 ▶ กลุ่มคำสั่งฟังก์ชันทางกราฟิกในเทอร์โบซี

1. ฟังก์ชันควบคุมทางด้านกราฟิก

closegraph	—	ยกเลิกการแสดงแบบกราฟิกโหมด
detectgraph	—	ตรวจสอบจอที่ใช้ว่าเป็นจอชนิดใด เพื่อเลือกใช้ไดรเวอร์ของแต่ละจอได้ถูกต้อง
grahpdefaults	—	กำหนดตัวแปรทางกราฟิกให้มีค่าตามเหมือนกับตอนที่เริ่มเข้ากราฟิกโหมด
_graphfreemem	—	ยกเลิกการจองหน่วยความจำในกราฟิกโหมด
_graphgetmem	—	จองหน่วยความจำที่ใช้ในกราฟิกโหมด
getgraphmode	—	ให้ค่าโหมดของจอที่ใช้อยู่ในปัจจุบัน ▶

getmoderange	—	ให้ค่าต่ำสุดและสูงสุดของโหมดที่ไดรเวอร์หนึ่ง
initgraph	—	ทำการติดตั้งระบบกราฟิกและเลือกโหมด
registerbgidriver	—	ทำการลิงค์ไดรเวอร์เข้าในระบบกราฟิก (ใช้เมื่อต้องการลิงค์ไดรเวอร์กับโปรแกรม)
restorecrtmode	—	กลับไปยังสกรีนโหมดเดิมก่อนการใช้คำสั่ง initgraph
setgraphbufsize	—	ระบุขนาดของกราฟิกบัฟเฟอร์
setgraphmode	—	เลือกกราฟิกโหมดและทำการเคลียร์จอและกำหนดตัวแปรทางกราฟิกเป็นค่า default

2. ฟังก์ชันการวาด และการระบาย

การวาด

arc	—	วาดส่วนของวงกลม
circle	—	วาดวงกลม
drawpoly	—	วาดรูปหลายเหลี่ยม
ellipse	—	วาดวงรี หรือส่วนของวงรี
getarccoords	—	ให้ค่าตำแหน่งต่าง ๆ ของการวาดรูป ส่วนของวงกลมหรือวงรีครั้งสุดท้าย
getaspectratio	—	ให้ค่า aspect ratio ที่กราฟิกโหมดปัจจุบัน
getlinesetting	—	ให้ค่า รูปแบบ, ลักษณะ, ความหนา ของเส้นตรงที่จะวาด
line	—	วาดเส้นโดยกำหนดพิกัดเริ่มต้นและสุดท้าย
linerel	—	วาดเส้นโดยเทียบกับตำแหน่งการวาดปัจจุบัน (Correrrt Position)
lineto	—	วาดเส้นโดยเริ่มจากตำแหน่งการวาดปัจจุบันไปยังตำแหน่งที่กำหนด
moveto	—	ย้ายตำแหน่งการวาดไปตำแหน่งที่กำหนด
moverel	—	ย้ายตำแหน่งการวาดไปโดยเทียบกับตำแหน่งการวาดปัจจุบัน
rectangle	—	วาดรูปสี่เหลี่ยม
setlinestyle	—	กำหนด ความหนาและรูปแบบของเส้นที่จะวาด

การระบาย

bar	—	วาดรูป bar กราฟและระบายสีและแบบลงบนกราฟ
bar3d	—	วาดรูป bar กราฟ 3 มิติและทำการระบายและแบบลงบนกราฟ
fillpoly	—	ระบายสีและแบบลงในรูปหลายเหลี่ยม
floodfill	—	ระบายสีและแบบโดยวิธีหาขอบ
getfillpattern	—	ให้ค่าลักษณะของการระบายที่ผู้ใช้กำหนดไว้
getfillsettings	—	ให้ข้อมูลเกี่ยวกับการระบายปัจจุบัน
pieslice	—	วาดรูปส่วนของ pie และระบายสีและแบบ
setfillpattern	—	กำหนดลักษณะการระบายโดยผู้ใช้จะกำหนดรูปแบบเอง
setfillstyle	—	เลือกลักษณะการระบาย

3. ฟังก์ชันย้ายข้อมูลกราฟิกและ viewport

cleardevice	—	ลบกราฟิกสกรีน
setactivepage	—	เลือกกราฟิกเพจที่จะวาด
setvisualpage	—	เลือกกราฟิกเพจที่จะใช้แสดงบนจอ
clearviewport	—	ยกเลิก viewport ปัจจุบัน
getviewsettings	—	ให้ค่าเกี่ยวกับ viewport ปัจจุบัน
setviewport	—	กำหนดตำแหน่งของ viewport
getimage	—	เก็บภาพแบบบิพแมตในบริเวณที่กำหนด
imagesize	—	ให้ค่าจำนวนไบต์ของขนาดที่ใช้เก็บภาพในบริเวณ 4 เหลี่ยมที่กำหนด
putimage	—	วาดภาพตามที่เก็บไว้ ณ ตำแหน่งที่กำหนด
getpixel	—	ให้ค่าสีของ pixel ที่ตำแหน่ง x, y
putpixel	—	พลอตจุดที่ตำแหน่ง x, y

4. ฟังก์ชันการเขียนตัวอักษร

gettextsettings	—	ให้ค่ารูปแบบ, ทิศทาง, ขนาดและการจัดวางของ font ปัจจุบัน

outtext	—	เขียนข้อความลงจอเริ่มที่ตำแหน่งการวาดปัจจุบัน
outtextxy	—	เขียนข้อความลงจอโดยเริ่มตามตำแหน่งที่กำหนด
registerbgifont	—	ทำการลิงค์ฟอนต์เข้าในระบบกราฟิก
settextjustify	—	เลือกการจัดวางของข้อความที่ใช้ใน outtext และouttextxy
settextstyle	—	เลือกรูปแบบ, ทิศทาง และขนาดของฟอนต์
setusercharsize	—	เลือกความกว้าง, ความสูงของ stroked fonts
textheight	—	ให้ค่าความสูงของข้อความเป็นพิกเซล
textwidth	—	ให้ค่าความกว้างข้อความเป็นพิกเซล

5. ฟังก์ชันควบคุมสี

getbkcolor	—	ให้ค่าสีพื้นปัจจุบัน
getcolor	—	ให้ค่าสีที่จะวาดปัจจุบัน
getmaxcolor	—	ให้ค่าของสีสูงสุดที่วาดได้ในโหมดปัจจุบัน
getpalette	—	ให้ค่า palette และขนาดของ palette ปัจจุบัน
setallpalette	—	เปลี่ยนสี palette ทั้งหมดตามที่ระบุ
setbkcolor	—	เลือกสีพื้นปัจจุบัน
setcolor	—	เลือกสีที่จะวาดปัจจุบัน
setpalette	—	เปลี่ยนสี palette หนึ่งสีตามที่ระบุ

6. ฟังก์ชันบ่งบอกความผิดพลาด

grapherrormsg	—	ให้ข้อความบ่งบอกความผิดพลาดตาม error code
graphresult	—	ให้ค่า errorcode สำหรับการทำงานกราฟิกคำสั่งสุดท้าย

หมายเหตุ นอกจากนี้ยังมีอีก 4 คำสั่งที่ยังไม่กล่าวถึงคือ

getmaxx	—	ให้ค่าพิกัดสูงสุดในแกน x ปัจจุบัน
getmaxy	—	ให้ค่าพิกัดสูงสุดในแกน y ปัจจุบัน
getx	—	ให้ค่าพิกัด Cursor Position ปัจจุบัน
gety	—	ให้ค่าพิกัด Cursor Position ปัจจุบัน

ในกลุ่มคำสั่งแรก ฟังก์ชัน initgraph เป็นฟังก์ชันที่สำคัญที่สุด และมักจะเป็นที่สงสัยของผู้ที่เริ่มใช้ ซึ่งมีรูปแบบดังนี้

```
void far initgraph (int far *graphdriver, int far
*graphmode char far *pathtodriver);
```

เราสามารถเรียกใช้ฟังก์ชันนี้ได้ใน 2 ลักษณะคือ

1. ให้ฟังก์ชันตรวจสอบจอเองว่าเป็นจอชนิดใดแล้วโหลดไดรเวอร์สำหรับจอนั้นมาใช้โดยจะเลือกสกรีนโหมดที่มีความละเอียดสูงสุดสำหรับจอนั้น

2. ระบุว่าจะใช้ไดเวอร์สำหรับจอใด จะทราบได้โดยใช้ฟังก์ชัน detectgraph เพื่อตรวจสอบจอแล้วจึงส่งผ่านค่าให้ฟังก์ชัน initgraph อีกที วิธีนี้จะดีกว่า

รูปที่ 2 การกำหนดค่าคงที่สำหรับ graphdriver

graphics _driver Constant	Numeric Value
DETECT	0
CGA	1
MCGA	2
EGA	3
EGA64	4
EGAMONO	5
RESERVED	6
HERCMONO	7
ATT400	8
VGA	9
PC 3270	10

วิธีแรก เนื่องจากเราสามารถเลือกสกรีนโหมดได้ แทนที่จะเป็นโหมดที่มีความละเอียดสูงสุด

การระบุ graphdriver ควรจะระบุโดยใช้ค่าคงที่ที่กำหนดไว้ก่อนแล้วใน graphics.h ซึ่งมีค่าต่าง ๆ ดังรูปที่ 2 ถ้า graphdrive มีค่าเป็น DETECT ฟังก์ชัน initgrap จะทำการ Detect จอเอง

ตัวแปร graphmode จะกำหนดกราฟิกโหมดที่เลือก ซึ่งแต่ละจอจะมีจำนวนสกรีนโหมดไม่เท่ากัน แสดงดังรูปที่ 3

ส่วน pathtodriver คือตัวแปรระบุว่า path ของไดรเวอร์ ไฟล์ ซึ่งมีนามสกุล .bgi อยู่ที่ใด ซึ่งก็อย่าลืมว่าถ้าไดรเวอร์ของเราอยู่ใน subdirectory ภาษาซีจะต้องเพิ่ม '/' อีกต่อหนึ่ง หรือถ้าส่งเป็น "" (null string) ก็จะหมายถึง Current drive

ตัวอย่างการใช้ initgraph แบบแรกโดยมีไดรเวอร์บน current drive จะเป็น

```
#include<graphics.h>
main()
{
    int graph_driver = DETECT;
    int graph_mode;
    .
    .
    .
    initgraph(&graph_driver,&graph_mode,
              "");
    .
    .
    .
    closegraph();
```

ถ้า initgraph ในลักษณะที่ 2 โดยถ้าเป็นจอ CGA ให้เลือกเป็น mode CGAC 0 และไดรเวอร์อยู่ที่ C:\TURBOC โปรแกรมจะมีลักษณะดังนี้

```
#include<graphics.h>
main()
{
    int g_driver,g_mode;
    .
    .
    .
    detectgraph(&g_driver,&g_mode);
    if(g_driver == CGA){
        g_mode = CGAC0;
    }
    initgraph(&g_driver,&g_mode,"c:\\turboc");
    .
    .
    .
    closegraph();
```

สำหรับฟังก์ชัน restorecrtmode จะทำการเปลี่ยนสกรีนโหมด จากกราฟิกโหมดไปยังเท็กซ์โหมดปกติจะใช้คู่กับ

graphics driver	graphics_modes	Column Value	× Row	Palette	Pages
CGA	CGAC0	0	320×200	C0	1
	CGAC1	1	320×200	C1	1
	CGAC2	2	320×200	C2	1
	CGAC3	3	320×200	C3	1
	CGAHI	4	640×200	2 color	1
MCGA	MCGAC0	0	320×200	C0	1
	MCGAC1	1	320×200	C1	1
	MCGAC2	2	320×200	C2	1
	MCGAC3	3	320×200	C3	1
	MCGAMED	4	640×200	2 color	1
	MCGAHI	5	640×480	2 color	1
EGA	EGALO	0	640×200	16 color	4
	EGAHI	1	640×350	16 color	2
EGA64	EGA64LO	0	640×200	16 color	1
	EGA64HI	1	640×350	4 color	1
EGA-MONO	EGAMONOHI	3	640×350	2 color	1*
	" "	3	" "	2 color	2**
HERC	HERCMONOHI	0	720×348	2 color	2
ATT400	ATT400C0	0	320×200	C0	1
	ATT400C1	1	320×200	C1	1
	ATT400C2	2	320×200	C2	1
	ATT400C3	3	320×200	C3	1
	ATT400MED	4	640×200	2 color	1
	ATT400HI	5	640×400	2 color	1
VGA	VGALO	0	640×200	16 color	2
	VGAMED	1	640×350	16 color	2
	VGAHI	2	640×480	16 color	1
PC3270	PC3270HI	0	720×350	2 color	1

*64K on EGAMONO card
**256K on EGAMONO card

รูปที่ 3 แสดงจำนวนสกรีนโหมดของแต่ละกราฟิกไดรเวอร์

ตัวอย่างโปรแกรม MINI BOARD

ฟังก์ชัน setgraphmode โดยสามารถกลับไปมาระหว่างเท็กซ์โหมดและกราฟิกโหมด แต่การใช้ฟังก์ชันคู่นี้โดด ๆ ในการเปลี่ยนโหมดนั้น จะทำให้รูปภาพที่วาดไว้จะถูกเคลียร์ ตัวแปรต่าง ๆ ในกราฟิกโหมดที่ทำการเซตค่าไว้จะถูกเซตกลับไปมีค่า default คือเหมือนกับตอน initgraph ดังนั้น ถ้าเราต้องการกลับไปที่เท็กซ์โหมดแล้วกลับมาที่กราฟิกโหมดโดยให้มีรูปและค่าตัวแปรทางกราฟิกเป็นดังเดิมจะต้องทำการเก็บรูปภาพนั้นและค่าของตัวแปรทางกราฟิกไว้ก่อนแล้วจึงใช้คำสั่ง restorecrtmode และเมื่อจะกลับสู่กราฟิกโหมด ก็เรียกใช้ฟังก์ชัน setgraphmode จากนั้นจึงทำการวาดรูปที่เก็บไว้และเซตตัวแปรทางกราฟิกให้เป็นอย่างเดิม

ในกลุ่มคำสั่งที่สองเป็นคำสั่งการวาดรูปกราฟิกพื้นฐานและยังมีคำสั่งที่เกี่ยวกับการวาดรูป bar กราฟและ pie กราฟ สามารถกำหนดรูปแบบ, ลักษณะและความหนาของเส้นที่วาด เลือกลักษณะและกำหนดลักษณะการระบายเองได้ สิ่งที่ควรระวังในการวาดรูปกราฟิกนี้ก็คือ การอ้างพิกัดในกราฟิกโหมด จะเป็นในลักษณะของ "world of offset" คือจุดเริ่มต้นจะอยู่มุมบนซ้ายของจอจะเป็นพิกัด 0,0 โดยค่าในแกน x จะเพิ่มจากซ้ายไปขวา ส่วนแกน y จะเพิ่มจากบนลงล่าง ซึ่งไม่เหมือนกันกับเท็กซ์โหมดจะเป็นลักษณะของ "world of count" ตัวอักษรเริ่มต้นของจออยู่มุมบนซ้ายเช่นกัน และเป็น row ที่ 1 และ column ที่ 1 ตัวอย่างเช่น ในจอ CGA mode CGAC0 ซึ่งมีความละเอียดเป็น 320×200 ดังนั้นมุมบนซ้ายและมุมล่างขวาของจอจะอ้างด้วยพิกัด 0,0 และ 319, 199 ตามลำดับ ไม่ใช้ 1,1 และ 320, 200 ถ้าเข้าใจผิดในเรื่องการอ้างพิกัดนี้อาจทำให้รูปที่วาดกลับหัว หรือรูปในบริเวณขอบมีการผิดพลาดได้

ในกลุ่มคำสั่งที่สาม ฟังก์ชันที่น่ากล่าวถึงก็คือฟังก์ชันที่เกี่ยวกับการแสดงผลกราฟิก โดยการเปลี่ยนเพจซึ่งมีจอ EGA, VGA และ Hercules ที่ซัพพอร์ตในเรื่อง multiple page นี้ ส่วนจออื่นจะมีเพียงเพจเดียว (ดูรูปที่ 3) ทำให้เราสามารถวาดรูปในอีกเพจหนึ่งในขณะที่แสดงผลที่อีกเพจหนึ่งได้ และสามารถไปประยุกต์ใช้กับโปรแกรมที่ต้องการวาดภาพจำนวนมากกว่าหนึ่งภาพพร้อม ๆ กัน และแสดงผลกลับไปมาระหว่างรูปได้อย่างรวดเร็ว แทนที่จะใช้วิธีเก็บภาพหนึ่งไว้แล้ววาดอีกภาพหนึ่งซึ่งช้ากว่าวิธีแรกมาก

คำสั่งที่น่าสนใจอีกอันหนึ่งคือ กลุ่มคำสั่งที่เกี่ยวกับ viewport viewport ก็เปรียบได้กับ window ในเท็กซ์โหมด โดย viewport จะกำหนดเป็นบริเวณ 4 เหลี่ยม โดยมีจุดเริ่มต้นตามที่กำหนด ซึ่งจะทำให้การอ้างพิกัดต่าง ๆ เทียบกับจุดเริ่มต้นของ viewport ทำให้สามารถประยุกต์การใช้งานแบบวินโดว์ได้ง่าย

คำสั่งอื่น ๆ ที่เกี่ยวกับการเก็บรูป และวาดรูปที่เก็บไว้ในบริเวณที่กำหนด ซึ่งถ้าใช้ในการย้ายรูปที่ไม่ใหญ่นักไปยังพิกัดอื่นก็มีความเร็วน่าพอใจทีเดียว แต่ถ้าใช้กับรูปใหญ่ ๆ เช่นเก็บรูปทั้งจอความเร็วก็ลดลง

ในกลุ่มคำสั่งที่สี่ เป็นคำสั่งที่เกี่ยวกับการเขียนตัวอักษรแบบต่าง ๆ ซึ่งแบ่งออกเป็น 2 ประเภท คือ

1. bit-mapped font โดยตัวอักษรแต่ละตัวจะกำหนดด้วยกลุ่มของจุดประกอบเป็นตัวอักษร

2. stroked font โดยตัวอักษรแต่ละตัวจะกำหนดชุดของเวกเตอร์ประกอบกันเป็นตัวอักษรแต่ละตัว

ซึ่งตัวอักษรแบบบิตแมตจะใช้ได้ดีเมื่ออักษรมีขนาดเล็กเพราะมีจำนวนจุดไม่มาก ทำให้การเขียนตัวอักษรด้วยการพล็อตได้อย่างรวดเร็ว ส่วนในแบบ stroked font เหมาะกับตัวอักษรที่มีขนาดใหญ่ เพราะตัวอักษรแทนด้วยชุดของเวกเตอร์ เมื่อจะย่อหรือขยายก็เปลี่ยนสเกลของตัวอักษร ทำให้มีความละเอียดสูงกว่าแบบบิตแมต นอกจากนี้ ข้อความที่เขียนมีฟอนต์ให้เลือกหลายรูปแบบ สามารถเลือกการจัดวางเป็นแบบจัดขอบขวา, ซ้าย หรือให้อยู่กลางของจุดอ้างอิง หรือเขียนในอยู่ในแนวดิ่ง แนวนอน

การเขียนตัวอักษรแบบ stroked font นั้น จะต้องมีไฟล์ นามสกุล .CHR อยู่ใน pathtodriver ที่เดียวกับไฟล์นามสกุล .bgi ดังนั้น ทุกครั้งที่ทำการ run โปรแกรม จะต้องมีไฟล์เหล่านี้อยู่ใน pathtodriver เสมอ มิฉะนั้นจะไม่สามารถเข้าสู่กราฟิกโหมดได้ ในกรณีที่ขาด [driver] .bgi ไปโปรแกรมจะรายงาน error massage บอกมาว่า BGI ไม่ได้ install เนื่องจากหาไฟล์นี้ไม่เจอ หรือถ้าเขียน stroked font ออกจอโดยไม่มี [file font] .CHR เทอร์โบซีก็จะเขียนเป็นตัวอักษรบิตแมตให้แทนซึ่งมีอยู่ในกราฟิกไลบรารีของเทอร์โบซีอยู่แล้ว

ในกรณีที่ต้องการรวม [driver] .bgi และ/หรือ [file font] .CHR เข้าไปในเอ็กซีคิวต์ไฟล์ทำได้โดยอาศัยยูทิลิตี้ ซึ่งเทอร์โบซีเตรียมไว้แล้วคือ bgiobj.exe โดยยูทิลิตี้นี้จะทำการเปลี่ยน [driver] .bgi และ [file font] .CHR เป็นนามสกุล .obj ซึ่งสามารถนำไปรวมกับกราฟิกไลบรารีหรือจะนำไปรวมในตอนลิงค์ก็ได้ การทำเช่นนี้จะทำให้เอ็กซิคิวต์ไฟล์ก็ได้มีขนาดใหญ่ขึ้น แต่ต้องเพิ่มคำสั่ง registerbgidriver เข้าไปในโปรแกรมเมื่อต้องการลิงค์ไดรเวอร์เข้ามา และต้องเพิ่มคำสั่ง registerbgifont เข้าไปในโปรแกรมเมื่อต้องการลิงค์ฟอนต์เข้ามาก่อนคำสั่ง initgraph

ในกลุ่มคำสั่งที่ห้าเกี่ยวกับการควบคุมสีพื้นที่และสีที่จะวาด ซึ่งจำนวนสีขึ้นอยู่กับชนิดของจอและโหมดที่ใช้งานอยู่

ในขณะนั้น โดยชุดของสีที่เรียกว่า palette สามารถเปลี่ยนสีใน palette ได้ กำหนดสีและสีพื้นที่จะวาดได้

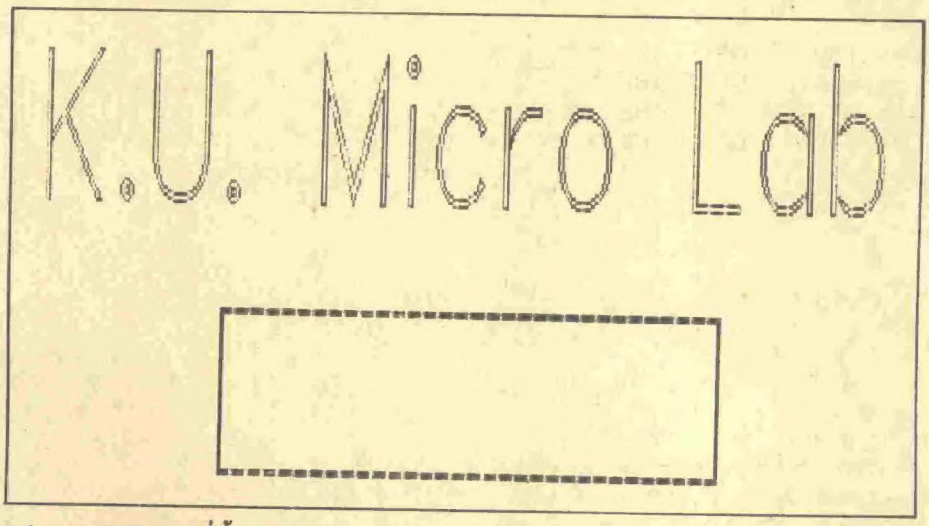
ในกลุ่มคำสั่งสุดท้ายมีประโยชน์ในการตรวจสอบและออกความผิดพลาดที่เกิดขึ้นในการทำงานของฟังก์ชันทางกราฟิก โดย graphresult จะให้ค่า error code ของคำสั่งกราฟิกคำสั่งสุดท้าย ซึ่งถ้าค่า error code น้อยกว่าศูนย์แสดงว่าเกิด error ขึ้น โดยมีค่าอยู่ในช่วง −15 ถึง −1 เมื่อส่งค่าไปให้ grapherrormsg ก็จะให้ข้อความรายงานเกี่ยวกับ error นั้นกลับมา โดยจะส่งค่ากลับมาเป็นพอยน์เตอร์ของคาแรกเตอร์

คำสั่งต่าง ๆ ทางกราฟิกนี้ ในดิสเกตต์ของเทอร์โบซีมีโปรแกรม bgidemo.C ซึ่งจะแสดงการทำงานฟังก์ชันของกราฟิกทั้งหมด เมื่อลอง RUN โปรแกรมนี้ก็จะเข้าใจคำสั่งต่าง ๆ ที่กล่าวมาแล้วได้ดีขึ้น ขอให้ลองดูนะครับ

MINI BOARD

เมื่อเราได้รู้คำสั่งต่าง ๆ ทางด้านกราฟิกแล้ว ลองมาดูโปรแกรมตัวอย่างสักตัวอย่างหนึ่ง โดยโปรแกรมนี้คล้ายกับการเขียน message ในโปรแกรม story board โดยโปรแกรมจะถาม ข้อความ, font และขนาดที่จะพิมพ์ลงจอในโหมดกราฟิก จากนั้นโปรแกรมจะตีกรอบลงจอเท่ากับขนาดข้อความในโหมดกราฟิกนั้น และสามารถเลื่อนกรอบไปณ ตำแหน่งที่เราต้องการให้เขียนลงจอ โดยใช้ปุ่ม up, down, left, right เมื่ออยู่ตำแหน่งที่พอใจแล้ว กดปุ่ม return ก็จะแสดงข้อความนั้นลงจอ จุดที่สำคัญในโปรแกรมนี้ก็คือการเลื่อนกรอบไปมาในกราฟิก โดยรูปภาพเดิมไม่เปลี่ยนแปลง เราจะใช้การเก็บภาพที่กรอบจะเขียนทับลงไปขึ้นมาก่อน จากนั้นจึงเขียนกรอบให้มีสีตรงข้ามกับสีเดิม เมื่อจะเลื่อนกรอบไปเราก็นำภาพของกรอบที่เก็บไว้เขียนทันกลับไปก่อนแล้วจึงเลื่อนกรอบไปยังตำแหน่งใหม่ นอกจากนี้ โปรแกรมนี้ยังสามารถ load, save, clear รูปที่เขียนไว้ ในตอนที่จะคอมพายล์ และลิงค์โปรแกรมนี้ อย่าลืมลิงค์กับกราฟิกไลบรารีด้วยนะครับ มิฉะนั้นจอ error ตอนลิงค์ ถ้าใช้ Integrated Environment ก็สร้างโปรเจคไฟล์ขึ้นมาเขียนชื่อโปรแกรมนี้ BOARD.C และตามด้วยชื่อไลบรารีที่จะลิงค์คือ graphics.lib ถ้าใช้ command line ก็เพิ่มชื่อ graphics.lib ที่ท้ายบรรทัด หรือถ้าใช้ graphics.lib บ่อย ๆ ก็นำไปรวมกับไลบรารีมาตรฐานที่ต้องลิงค์อยู่แล้ว ตัวอย่างเช่น ในกรณีที่คอมพายล์แบบ small model ก็อาจรวมกราฟิกไลบรารีกับ CS.lib เป็น CS.lib ตัวเดียว ทำให้สะดวกขึ้นไม่ต้องใช้โปรเจคไฟล์หรือเขียนไลบรารีที่จะลิงค์เพิ่มเข้าไป

จะเห็นได้ว่าถ้าเราพัฒนาโปรแกรมนี้ต่อไปโดยเก็บรูปที่ต้องการลงในไฟล์หลาย ๆ ไฟล์แล้ว จะมีโปรแกรมส่วนที่นำไฟล์มาแสดงทีละรูป ทีละรูป เราก็จะได้ story board มาใช้งานได้แล้ว และยิ่งไปกว่านั้นถ้าเราเพิ่มส่วนของ outtext ที่เป็นภาษาไทยเข้าไป เราก็จะได้โปรแกรม Thai story board หวังว่าบทความนี้คงมีส่วนช่วยให้ผู้อ่านได้รับไอเดียเกี่ยวกับกราฟิกพอสมควรนะครับ



รูปแสดงตัวอักษรที่ขึ้นบนจอและกรอบของตัวอักษรใหม่

รูปแสดงตัวอักษรที่เขียน

```
Listing of BOARD.C
File updated on :  9 October 1988  Time 14:11:50

 1: /*                                K.U. Micro Lab                          */
 2: /*     Program Mini Board                                                 */
 3: /*          input : message,font,size                                     */
 4: /*          output: message in graphics mode                              */
 5:
 6: #include<graphics.h>
 7: #include<stdio.h>
 8: #include<conio.h>
 9: #include<stdlib.h>
10: #include<alloc.h>
11:
12: enum BOOLEAN  {FALSE,TRUE};
13: enum KEY_TYPE  {RETURN,UP,DOWN,LEFT,RIGHT};
14: enum DRAW_MESSASS_STATUS {OVER_SCREEN,OK};
15:
16: struct frame_type {
17:     void *line1;        /* TOP    OF FRAME */
18:     void *line2;        /* LEFT   OF FRAME */
19:     void *line3;        /* BOTTOM OF FRAME */
20:     void *line4;        /* RIGHT  OF FRAME */
21: };
22:
23: main()
24: {
25:     int g_driver = DETECT,g_mode;
26:     void *picture;
27:     unsigned int pic_size;
28:     int loop,option;
29:
30:     initgraph(&g_driver,&g_mode,"c:\\turboc");
31:     pic_size = imagesize(0,0,getmaxx(),getmaxy());
32:     picture = (void *) malloc(pic_size);
33:     memset(picture,'\x00',pic_size);
34:     restorecrtmode();
35:     loop = TRUE;
36:     do {
37:         menu(&option);
38:         switch(option) {
39:             case 1: draw_picture(picture,g_mode);          break;
40:             case 2: view_picture(picture,g_mode);          break;
41:             case 3: clear_picture(picture,g_mode);         break;
42:             case 4: load_picture(picture,pic_size,g_mode); break;
43:             case 5: save_picture(picture,pic_size);        break;
44:             case 6: loop = FALSE;                          break;
45:         }
46:     } while (loop);
47:     closegraph();
48: }
49:
50: menu(int *option)
51: {
52:     char line[90];
53:
```

```
54:     *option = 0;
55:     while ((*option >6)||(*option < 1)) {
56:         clrscr();
57:         gotoxy(1,3);  printf("                                   MINI BOARD");
58:         gotoxy(1,6);  printf("            1. Draw  Message");
59:         gotoxy(1,7);  printf("            2. View  Picture");
60:         gotoxy(1,8);  printf("            3. Clear Picture");
61:         gotoxy(1,9);  printf("            4. Load  Picture");
62:         gotoxy(1,10); printf("            5. Save  Picture");
63:         gotoxy(1,11); printf("            6. EXIT");
64:         gotoxy(1,14); printf("     SELECT :");
65:         gets(line);
66:         sscanf(line,"%d",option);
67:     }
68: }
69:
70: alloc_frame(int x1,int y1,int x2,int y2,struct frame_type *frame)
71: {
72:     frame->line1 = (void *) malloc(imagesize(x1,y1,x2,y1+1));
73:     frame->line2 = (void *) malloc(imagesize(x1,y1,x1+1,y2));
74:     frame->line3 = (void *) malloc(imagesize(x1,y2-1,x2,y2));
75:     frame->line4 = (void *) malloc(imagesize(x2-1,y1,x2,y2));
76: }
77: get_frame(int x1,int y1,int x2,int y2,struct frame_type *frame)
78: {
79:     getimage(x1,y1,x2,y1+1,frame->line1);
80:     getimage(x1,y1,x1+1,y2,frame->line2);
81:     getimage(x1,y2-1,x2,y2,frame->line3);
82:     getimage(x2-1,y1,x2,y2,frame->line4);
83: }
84:
85: put_frame(int x1,int y1,int x2,int y2,struct frame_type *frame,int option)
86: {
87:     putimage(x1,y1,frame->line1,option);
88:     putimage(x1,y1,frame->line2,option);
89:     putimage(x1,y2-1,frame->line3,option);
90:     putimage(x2-1,y1,frame->line4,option);
91: }
92:
93: free_frame(struct frame_type *frame)
94: {
95:     free(frame->line1);
96:     free(frame->line2);
97:     free(frame->line3);
98:     free(frame->line4);
99:     free(frame);
100: }
101:
102: int get_key()
103: {
104:     int loop;
105:     int key;
106:     static int buffer;
107:
108:     loop = TRUE;
109:     do {
```

```
110:         if((key=getch())== 0) {
111:             key = getch();
112:             switch(key) {
113:                 case 72 : buffer = UP; loop = FALSE;     break;
114:                 case 80 : buffer = DOWN; loop = FALSE;   break;
115:                 case 75 : buffer = LEFT; loop = FALSE;   break;
116:                 case 77 : buffer = RIGHT; loop = FALSE;  break;
117:             }
118:         }else
119:             if (key==13) {
120:                 buffer = RETURN;
121:                 loop = FALSE;
122:             }
123:     } while(loop);
124:     return(buffer);
125: }
126:
127: move_frame(int key,int *x,int *y,int text_width,int text_height)
128: {
129:     switch(key) {
130:         case   UP : --*y;
131:                     if (*y < 0)
132:                         ++*y;
133:                     break;
134:         case DOWN : ++*y;
135:                     if (*y+text_height > getmaxy())
136:                             --*y;
137:                     break;
138:         case LEFT : --*x;
139:                     if (*x < 0)
140:                         ++*x;
141:                     break;
142:         case RIGHT: ++*x;
143:                     if (*x+text_width > getmaxx())
144:                         --*x;
145:                     break;
146:     }
147: }
148:
149: int draw_message(char *message,int size,int font,void *picture)
150: {
151:     int text_width,text_height;
152:     int x,y;
153:     struct frame_type *frame;
154:     int key;
155:
156:     settextstyle(font,HORIZ_DIR,size);
157:     text_width = textwidth(message);
158:     text_height = textheight(message);
159:     x = (getmaxx() +1 - text_width) / 2;  /* SET FRAME TO CENTER OF SCREEN */
160:     y = (getmaxy() +1 - text_height) / 2;
161:     if ( (x<0)||(y<0) ) {                 /* CHECK MESSAGE SIZE FOR SCREEN */
162:         return(OVER_SCREEN);
163:     } else {
164:         putimage(0,0,picture,COPY_PUT);
165:         frame = (struct frame_type *) malloc(sizeof(struct frame_type));
```

```
166:          alloc_frame(x,y,x+text_width,y+text_height,frame);
167:          get_frame(x,y,x+text_width,y+text_height,frame);
168:          put_frame(x,y,x+text_width,y+text_height,frame,NOT_PUT);
169:          while((key = get_key()) != RETURN) {
170:              put_frame(x,y,x+text_width,y+text_height,frame,COPY_PUT);
171:              move_frame(key,&x,&y,text_width,text_height);
172:              get_frame(x,y,x+text_width,y+text_height,frame);
173:              put_frame(x,y,x+text_width,y+text_height,frame,NOT_PUT);
174:          }
175:          put_frame(x,y,x+text_width,y+text_height,frame,COPY_PUT);
176:          free_frame(frame);
177:          outtextxy(x,y,message);
178:          return(OK);
179:      }
180: }
181:
182: read_message(char *message,int *size,int *font)
183: {
184:      char line[90];
185:
186:      clrscr();
187:      gotoxy(1,1); printf("                          DRAW MASSAGE");
188:      gotoxy(1,5); printf("       message :");
189:      gets(message);
190:      gotoxy(1,6); printf("       size    :");
191:      gets(line);
192:      sscanf(line,"%d",size);
193:      gotoxy(1,7); printf("       Font");
194:      gotoxy(1,8); printf("          0.Bit-mapped");
195:      gotoxy(1,9); printf("          1.Triplex");
196:      gotoxy(1,10);printf("          2.Small");
197:      gotoxy(1,11);printf("          3.Sansserif");
198:      gotoxy(1,12);printf("          4.Gothic");
199:      gotoxy(1,15);printf("        SELECT FONT:");
200:      gets(line);
201:      sscanf(line,"%d",font);
202: }
203:
204: draw_picture(void * picture,int g_mode)
205: {
206:      char message[200];
207:      int size,font;
208:
209:      read_message(message,&size,&font);
210:
211:      setgraphmode(g_mode);
212:      if (draw_message(message,size,font,picture) != OVER_SCREEN) {
213:          getimage(0,0,getmaxx(),getmaxy(),picture);
214:          getch();
215:          restorecrtmode();
216:      } else {
217:          restorecrtmode();
218:          gotoxy(1,5); printf("              message too big");
219:          getch();
220:      }
221: }
```

```
222:
223: view_picture(void *picture,int g_mode)
224: {
225:     setgraphmode(g_mode);
226:     putimage(0,0,picture,COPY_PUT);
227:     getch();
228:     restorecrtmode();
229: }
230:
231: clear_picture(void *picture,int g_mode)
232: {
233:     setgraphmode(g_mode);
234:     getimage(0,0,getmaxx(),getmaxy(),picture);
235:     restorecrtmode();
236: }
237:
238: load_picture(void *picture,int pic_size,int g_mode)
239: {
240:     FILE *fvar;
241:     char line[90];
242:     char filename[30];
243:
244:     clrscr();
245:     gotoxy(1,1); printf("                                LOAD PICTURE");
246:     gotoxy(1,5); printf("           Load Picture From File :");
247:     gets(line);
248:     sscanf(line,"%s",filename);
249:     if( (fvar = fopen(filename,"rb")) != NULL) {
250:         gotoxy(1,7); printf(" please wait !");
251:         fread(picture,pic_size,1,fvar);
252:         fclose(fvar);
253:         view_picture(picture,g_mode);
254:     } else {
255:         gotoxy(1,9); printf("       file %s not found",filename);
256:         getch();
257:     }
258: }
259:
260: save_picture(void *picture,int pic_size)
261: {
262:     FILE *fvar;
263:     char line[90];
264:     char filename[30];
265:
266:     clrscr();
267:     gotoxy(1,1); printf("                                SAVE PICTURE");
268:     gotoxy(1,5); printf("           Save Picture to File :");
269:     gets(line);
270:     sscanf(line,"%s",filename);
271:     fvar = fopen(filename,"wb");
272:     gotoxy(1,7); printf(" please wait !");
273:     fwrite(picture,pic_size,1,fvar);
274:     fclose(fvar);
275: }
276:
```